Both papers we are going to talk about in detail focus on automatic DL model design, trying to alleviate problems of previous approaches on the NASNet search space defined in [Zop+17].

To mention some of those previous papers, both [Zop+17] and [Rea+18] use a very similar approach – they sample better and better architectures from a high-level optimizer (RNN controller for [Zop+17] or an aging tournament selection EA favoring younger genotypes and thus promoting exploration for [Rea+18]) based on full converged training on a proxy, CIFAR-10 dataset, later using obtained cells on ImageNet.

These approaches, though providing great-performing models, ale not very efficient in terms of training, because of the huge number of full-convergence candidate model trainings. Just consider 2000 GPU days of reinforcement learning (RL) for [Zop+17] or 3150 GPU days of evolution for [Rea+18].

Another interesting approach that tries to lower the computational resources needed, a sequential model-based optimization (SMBO) strategy [Liu+17], searches for models of gradually increasing complexity while also learning a fast surrogate model that can choose promising extensions of those gradually-built models to guide the algorithm through the search space. This approach is 8 times faster than the RNN one.

The following papers lower the computational demands much further while preserving performance to a reasonable extent.

1 DARTS: Differentiable Architecture Search [LSY18]

DARTS address searching over a discrete set of candidate architectures by relaxing the search space to be continuous (they eventually get to the NASNet space again). Based on that, algorithm is guided through the search space by gradient descent (1.5 GPU days!).

"The data efficiency of gradient-based optimization, as opposed to inefficient black-box search, allows DARTS to achieve competitive performance with the state of the art using orders of magnitude less computation resources."

Using NASNet notation, they search for convolutional or recurrent cells that make up the whole architecture. They formalize the cell in the following way:

"A cell is a directed acyclic graph consisting of an ordered sequence of N nodes. Each node $x^{(i)}$ is a latent representation (e.g. a feature map in convolutional networks) and each directed edge (i, j) is associated with some operation $o^{(i,j)}$ that transforms $x^{(i)}$. We assume the cell to have two input nodes and a single output node. For convolutional cells, the input nodes are defined as the cell outputs in the previous two layers. For recurrent cells, these are defined as the input at the current step and the state carried from the previous step. The output of the cell is obtained by applying a reduction operation (e.g. concatenation) to all the intermediate nodes.

Each intermediate node is computed based on all of its predecessors:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$$

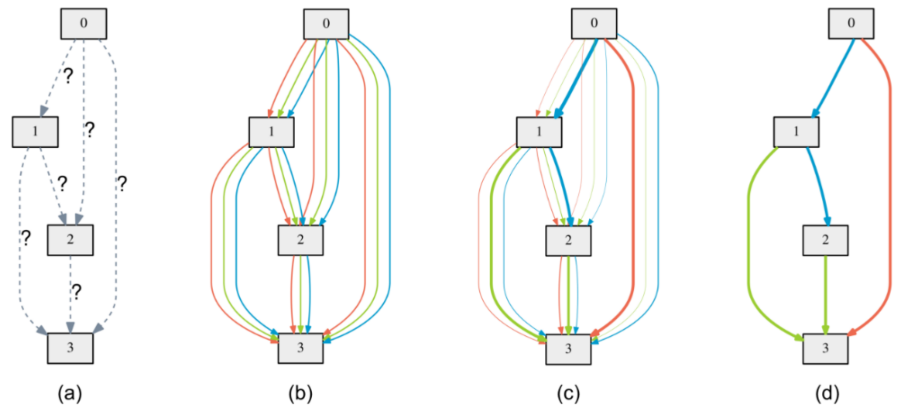


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

Let $\mathcal{O}$ be a set of candidate operations (e.g., convolution, max pooling, zero=no connection) where each operation represents some function $o(\cdot)$ to be applied to $x^{(i)}$. To make the search space continuous, they relax the categorical choice of a particular operation to a softmax over all possible operations:

$$\bar{o}^{(i,j)} = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

where the operation mixing weights for a pair of nodes (i, j) are parameterized by a vector $\alpha^{(i,j)}$ of dimension $|\mathcal{O}|$. The task of architecture search then reduces to learning a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$, as illustrated in Fig. 1. At the end of search, a discrete architecture can be obtained by replacing each mixed operation $\bar{o}^{(i,j)}$ with the most likely operation, i.e., $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}} \alpha_o^{(i,j)}$ (note: they do not do exactly this, more about that later). In the following, we refer to α as the (encoding of the) architecture.

Denote by $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ the training and the validation loss, respectively. Both losses are determined not only by the architecture α, but also the weights w in the network. The goal for architecture search is to find α^* that minimizes the validation loss $\mathcal{L}_{val}(w^*, \alpha^*)$, where the weights w^* associated with the architecture are obtained by minimizing the training loss $w^* = \text{argmin}_w \mathcal{L}_{train}(w, \alpha^*)$.

This implies a bilevel optimization problem with α as the upper-level variable and w as the lower-level variable:

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha) \tag{1}$$

$$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \, \mathcal{L}_{train}(w, \alpha) \,.\text{"} \tag{2}$$

Further, authors simplify composite gradient $\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha)$ to two approximate forms that are simpler to evaluate (first and second order), their experimentation uses both.

First order approximation ("The idea is to approximate $w^*(\alpha)$ by adapting w using only a single training step, without solving the inner optimization (equation 2) completely by training until convergence."):

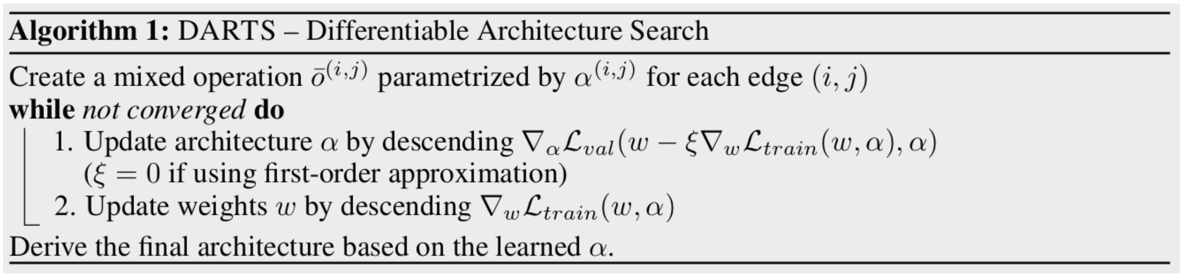
$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha) \,,$$

second order approximation:

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}(w', \alpha) - \xi \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\epsilon} \,,$$

where $w^\pm = w \pm \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$. For the second fraction term, that approximates more complicated second order gradient yielded by the chain rule, authors note: "Evaluating the finite difference requires only two forward passes for the weights and two backward passes for α, and the complexity is reduced from $O(|\alpha||w|)$ to $O(|\alpha| + |w|)$."

First order approximation algorithm:



```
Algorithm 1: DARTS – Differentiable Architecture Search
Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge $(i, j)$
while not converged do
    1. Update architecture $\alpha$ by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$
       ($\xi = 0$ if using first-order approximation)
    2. Update weights $w$ by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
Derive the final architecture based on the learned $\alpha$.
```

Finally, they claim: "While we are not currently aware of the convergence guarantees for our optimization algorithm, in practice it is able to reach a fixed point with a suitable choice of ξ." They also mention: "A simple working strategy is to set ξ equal to the learning rate for w's optimizer.

To be comparable with previous approaches, they derive discrete architecture in a bit different way than argmax (over every color triplet in Figure. 1, as a concrete example). For each node, they keep 2 best operations from all coming into the node for convolutional cells, whereas only 1 in case of recurrent cells (excluding zero operations from obvious reasons in both cases).

Pros:

- I like the idea a lot, its very simple and leads to well performing architectures (not exactly state-of-the-art), but 1 000 fold faster than previous approaches.

- The article's critical part is understandable and nicely written.
- Because of the points above and nice experimentation notes, one should be easily able to reproduce the experiments without the necessity of having Google's hardware options (1.5 GPU days).

Cons:

- They mention Zoph's articles several times including the one with NASNet space definition, but they do not elaborate search space section well enough in the beginning. It is not exactly the same as NASNet (it's more general, to be precise), but after all, they produce final architecture from NASNet subspace, to be able to compare it.
- Further to the above, Figure one and $\text{argmax}_{o \in \mathcal{O}} \alpha_o^{(i,j)}$ mislead readers from what they actually did. I would prefer following real experimentation and mention about bigger expressiveness of their search space and the possibility to use per-relaxed-operation argmax for all relaxed-operations coming to a node.

2 ENAS: Efficient Neural Architecture Search via Parameter Sharing [Pha+18]

Although introduced earlier than DARTS, this article presents even a bit faster search algorithm (0.5 GPU days). Produced models are comparable with DARTS, but bigger (cca 4.4 M parameters comparing to 3.3 M parameters).

Authors address the problem with training submodels generated by a controller to full convergence via utilizing the fact that all such models can be seen as a subgraph of a large computational DAG.

This observation led them to an idea of using the large computational graph 'as is' – meaning that subgraph models use its weights. Thus, they still use a controller network (RNN trained by gradient policy algorithm guided by the expected reward on a validation set, more on that later) achieving the search speed up by utilizing those shared parameters.

"The idea has apparent complications, as different child models might utilize their weights differently, but was encouraged by previous work on transfer learning and multitask learning."

"Intuitively, ENAS's DAG is the superposition of all possible child models in a search space of NAS, where the nodes represent the local computations and the edges represent the flow of information. The local computations at each node have their own parameters, which are used only when the particular computation is activated (note: they are misconceptual here, sometimes the associate parameters with edges). Therefore, ENAS's design allows parameters to be shared among all child models, i.e. architectures, in the search space."

Their notation is different from DARTS as they have no unifying space for recurrent and convolutional networks. (In DARTS, nodes are latent representations and all computations are done on the edges (thus unifying RNN and CNN architecture searches).) It can be seen on Figure 2 vs. Figure 3 with 4; they actually present three separate search spaces, RNN space, CNN architecture space, NASNet cell space that behaves very similarly from high level points of view. They are similar in a way that authors do not enable bigger search spaces than size

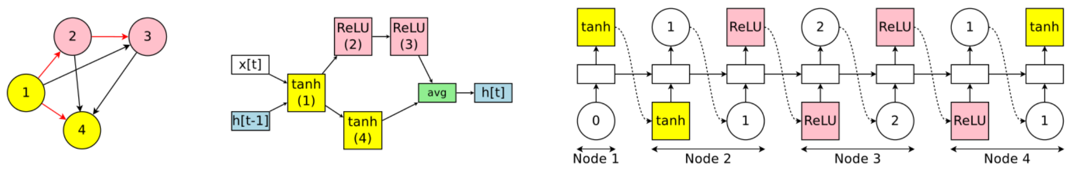


Figure 2: An example of a recurrent cell in our search space with 4 computational nodes. Left: The computational DAG that corresponds to the recurrent cell. The red edges represent the flow of information in the graph. Middle: The recurrent cell. Right: The outputs of the controller RNN that results in the cell in the middle and the DAG on the left. Note that nodes 3 and 4 are never sampled by the RNN, so their results are averaged and are treated as the cell's output.

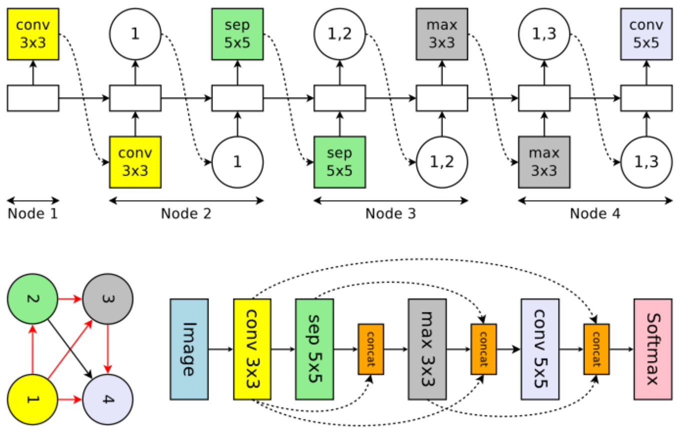


Figure 3: An example run of a recurrent cell in our search space with 4 computational nodes, which represent 4 layers in a convolutional network. Top: The output of the controller RNN. Bottom Left: The computational DAG corresponding to the network's architecture. Red arrows denote the active computational paths. Bottom Right: The complete network. Dotted arrows denote skip connections.

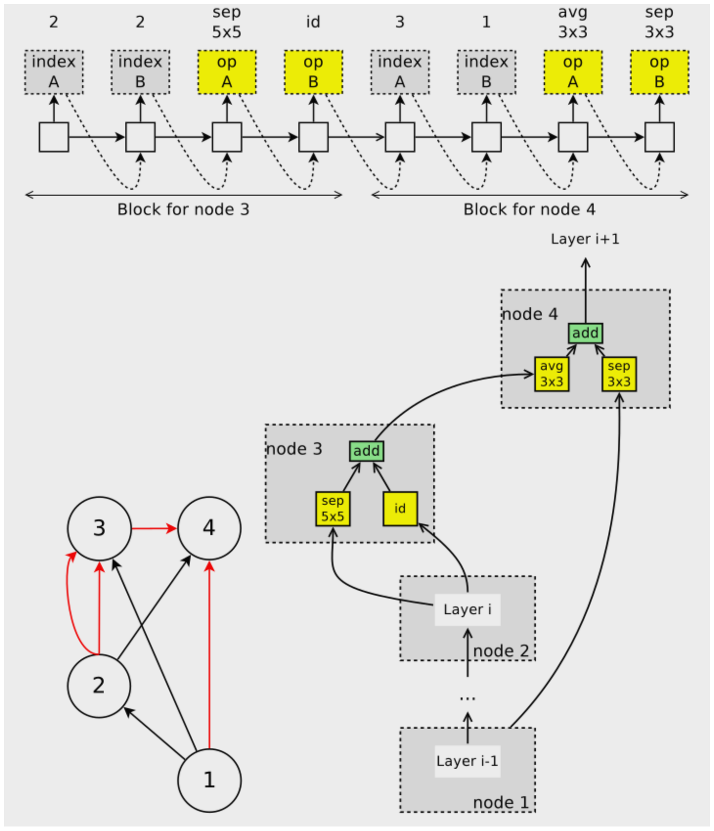


Figure 4: An example run of the controller for our search space over convolutional cells. Top: the controller's outputs. In our search space for convolutional cells, node 1 and node 2 are the cell's inputs, so the controller only has to design node 3 and node 4. Bottom Left: The corresponding DAG, where red edges represent the activated connections. Bottom Right: the convolutional cell according to the controller's sample.

of architecture they want to produce. Thus, when the controller finishes with generating an architecture, all nodes of the large computational graphs were processed. The controller's work is also similar in all cases, at each step (at each node), we sample some number of predecessor nodes and sample operation taken on a combination of sampled inputs.

Here, for RNNs, nodes are activation functions and edges between nodes $(i, j), i < j$ represent models-shared parameter matrices $\mathbf{W}_{(i,j)}$.

For CNN networks, nodes mean computations and edges just flow of information. Each node has associated its own weights for each of members from the predefined set of possible operations (convs, pools). Concatenations are inserted as needed given the number of predecessors.

For CNN cells, nodes are even more complicated as they contain always two operations and addition.

Finally, **training**. "Controller network is an LSTM with 100 hidden units. This LSTM samples decisions via softmax classifiers, in an autoregressive fashion: the decision in the previous step is fed as input embedding into the next step. At the first step, the controller network receives an empty embedding as input.

In ENAS, there are two sets of learnable parameters: the parameters of the controller LSTM, denoted by θ, and the shared parameters of the child models, denoted by ω. The training procedure of ENAS consists of two interleaving **minimization** phases. The first phase trains ω, the shared parameters of the child models, on a whole pass through the training data set. The second phase trains θ, the parameters of the controller LSTM, for a fixed number of steps.

Training the shared parameters ω of the child models. In this step, we fix the controller's policy $\pi(m;\theta)$ (m represents an architecture here) and perform stochastic gradient descent (SGD) on ω to minimize the expected loss function $\mathbb{E}_{m\sim\pi}[\mathcal{L}(m;\omega)]$. Here, $\mathcal{L}(m;\omega)$ is the standard cross-entropy loss, computed on a minibatch of training data, with a model m sampled from $\pi(m;\theta)$. The gradient is computed using the Monte Carlo estimate

$$\nabla_\omega \mathbb{E}_{m\sim\pi(m;\theta)}[\mathcal{L}(m;\omega)] \approx \nabla_\omega \frac{1}{M}\sum_{i=1}^{M}\mathcal{L}(m_i;\omega)\,,$$

where m_i's are sampled from $\pi(m;\theta)$ as described above. We find that $M=1$ works just fine, i.e. one can update ω using the gradient from any single model m sampled from $\pi(m;\theta)$. As mentioned, we train ω during an entire pass through the training data.

Training the controller parameters θ. In this step, we fix ω and update the policy parameters θ, aiming to maximize the expected reward $\mathbb{E}_{m\sim\pi(m;\theta)}[\mathcal{R}(m;\omega)]$. We employ the Adam optimizer, for which the gradient is computed using REINFORCE, with a moving average baseline to reduce variance. The reward $\mathcal{R}(m;\omega)$ is computed on the validation set, rather than on the training set, to encourage ENAS to select models that generalize well rather than models that overfit the training set well. In language model experiment, the reward function is $c/\text{valid}_\text{p}\text{pl}$, where the perplexity is computed on a minibatch of validation data. In image classification experiments, the reward function is the accuracy on a minibatch of validation images.

Deriving Architectures. Having a trained ENAS model, we first sample several models from the trained policy $\pi(m;\theta)$. For each sampled model, we compute its reward on a single minibatch sampled from the validation set. We then take only the model with the highest reward to re-train from scratch. It is possible to improve our experimental results by training all the sampled models from scratch and selecting the model with the highest performance on a separated validation set, as done by other works. However, our method yields similar performance whilst being much more economical."

Pros:

- When compared to DARTS, they use more explanatory figures.
- After all, information in the article are pretty elaborate.
- They also consider a question if ENAS produces empirically good models because of the search space definitions or because it is a good search algorithm. They answer this by comparing with random search and **ENAS with disabled controller updates**, which is nice.

Cons:

- The structure. They jump from one search space to another with a quick jump to the training process. They also use bibliography with broken hypertext links.
- Too wordy. It was a bit harder to read it because combination of jumping and repeating several very similar things did not help.

Bibliography

[Liu+17] Chenxi Liu et al. "Progressive Neural Architecture Search". In: *CoRR* abs/1712.00559 (2017). arXiv: `1712.00559`. URL: `http://arxiv.org/abs/1712.00559`.

[Zop+17] Barret Zoph et al. "Learning Transferable Architectures for Scalable Image Recognition". In: *CoRR* abs/1707.07012 (2017). arXiv: `1707.07012`. URL: `http://arxiv.org/abs/1707.07012`.

[LSY18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. "DARTS: Differentiable Architecture Search". In: *CoRR* abs/1806.09055 (2018). arXiv: `1806.09055`. URL: `http://arxiv.org/abs/1806.09055`.

[Pha+18] Hieu Pham et al. "Efficient Neural Architecture Search via Parameter Sharing". In: *CoRR* abs/1802.03268 (2018). arXiv: `1802.03268`. URL: `http://arxiv.org/abs/1802.03268`.

[Rea+18] Esteban Real et al. "Regularized Evolution for Image Classifier Architecture Search". In: *CoRR* abs/1802.01548 (2018). arXiv: `1802.01548`. URL: `http://arxiv.org/abs/1802.01548`.